Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по лабораторной работе №7

Тема: «Шаблоны классов.»

Выполнил

Студент группы РИС-22-2б

Максимов А.А.

Проверил доц. Кафедры ИТАС

Полякова О.А.

Пермь 2023

# Постановка задачи

# Определить шаблон класса-контейнера (см. лабораторную работу №6).

# Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.

# Перегрузить операции, указанные в варианте.

# Инстанцировать шаблон для стандартных типов данных (int, float, double).

# Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы стандартных типов данных.

# Реализовать пользовательский класс (см. лабораторную работу №3).

# Перегрузить для пользовательского класса операции ввода-вывода.

# Перегрузить операции необходимые для выполнения операций контейнерного класса.

# Инстанцировать шаблон для пользовательского класса.

# Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы пользовательского класса.

ВАРИАНТ 15:

Класс- контейнер СПИСОК с ключевыми значениями типа int.

Реализовать операции: [] – доступа по индексу; int() – определение размера списка; \* вектор – умножение элементов списков a[i]\*b[i];

Пользовательский класс Pair (пара чисел).

Пара должна быть представлено двумя полями: типа int для первого числа и типа double для второго.

Первое число при выводе на экран должно быть отделено от второго числа двоеточием.

**Контрольные вопросы**

*1. В чем смысл использования шаблонов?*

Смысл использования шаблонов заключается в возможности создания универсальных функций и классов, которые могут работать с различными типами данных без необходимости написания отдельного кода для каждого типа.

*2. Каковы синтаксис/семантика шаблонов функций?*

Синтаксис шаблонов функций выглядит следующим образом:

template <typename T>

void functionname(T parameter) {

// код функции

}

*3. Каковы синтаксис/семантика шаблонов классов?*

Синтаксис шаблонов классов выглядит следующим образом:

template <typename T>

class ClassName {

// компонентные функции и переменные класса

};

*4. Что такое параметры шаблона функции?*

Параметры шаблона класса и функции - это специальные типы данных, которые используются внутри шаблона для работы с различными типами данных. Они задаются при объявлении шаблона и могут быть любым типом данных, включая другие шаблоны.

*5. Перечислите основные свойства параметров шаблона функции.*

Основные свойства параметров шаблона функции: они являются универсальными и могут принимать любой тип данных, они могут быть переданы в качестве аргументов функции, они могут использоваться для создания универсальных алгоритмов.

*6. Как записывать параметр шаблона?*

Параметр шаблона записывается в угловых скобках после ключевого слова template:

template <typename T>

void functionname(T parameter) {

// код функции

}

*7. Можно ли перегружать параметризованные функции?*

Да, параметризованные функции могут быть перегружены.

*8. Перечислите основные свойства параметризованных классов.*

Основные свойства параметризованных классов: они могут работать с различными типами данных, они могут содержать компонентные функции и переменные, они могут наследоваться от других классов.

*9. Все ли компонентные функции параметризованного класса являются параметризованными?*

Нет, не все компонентные функции параметризованного класса являются параметризованными. Некоторые из них могут работать только с определенными типами данны

*10. Являются ли дружественные функции, описанные в параметризованном классе, параметризованными?*

Да, дружественные функции, описанные в параметризованном классе, также могут быть параметризованными.

*11. Могут ли шаблоны классов содержать виртуальные компонентные функции?*

Да, шаблоны классов могут содержать виртуальные компонентные функции.

*12. Как определяются компонентные функции параметризованных классов вне определения шаблона класса?*

Компонентные функции параметризованных классов могут быть определены вне определения шаблона класса следующим образом:

template <typename T>

class ClassName {

// компонентные функции и переменные класса

};

template <typename T>

void ClassName<T>::functionname() {

// код функции

}

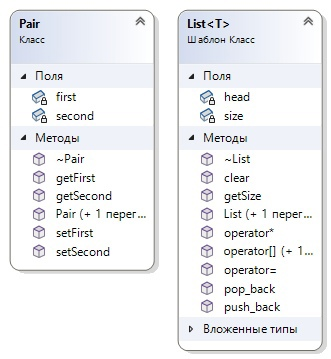
*13. Что такое инстанцирование шаблона?*

Инстанцирование шаблона - это процесс создания экземпляра класса или функции на основе шаблона с конкретными параметрами.

*14. На каком этапе происходит генерирование определения класса по шаблону?*

Генерирование определения класса по шаблону происходит во время компиляции программы. Компилятор создает новый класс или функцию для каждого уникального набора параметров шаблона, которые используются в программе.

UML – таблица



Код программы

**pair.h**

#ifndef PAIR\_H

#define PAIR\_H

#include <iostream>

class Pair {

public:

Pair() : first(0), second(0.0) {}

Pair(int f, double s) : first(f), second(s) {}

~Pair() {}

int getFirst() const { return first; }

double getSecond() const { return second; }

void setFirst(int f) { first = f; }

void setSecond(double s) { second = s; }

private:

int first;

double second;

};

#endif // PAIR\_H

**pair.cpp**

#include "pair.h"

// Перегрузка оператора ввода для класса Pair

std::istream& operator>>(std::istream& is, Pair& p) {

int f;

double s;

char colon;

is >> f >> colon >> s;

if (colon == ':') {

p.setFirst(f);

p.setSecond(s);

}

else {

is.setstate(std::ios::failbit);

}

return is;

}

// Перегрузка оператора вывода для класса Pair

std::ostream& operator<<(std::ostream& os, const Pair& p) {

os << p.getFirst() << ":" << p.getSecond();

return os;

}

**list.h**

#ifndef LIST\_H

#define LIST\_H

#include "pair.h"

#include <iostream>

template <typename T>

class List

{

private:

struct Node

{

T data;

Node\* next;

Node(const T& data, Node\* next = nullptr) : data(data), next(next) {}

};

Node\* head;

int size;

public:

List() : head(nullptr), size(0) {}

List(const List<T>& other) : head(nullptr), size(0) { \*this = other; }

~List() { clear(); }

List<T>& operator=(const List<T>& other)

{

if (this != &other)

{

clear();

for (Node\* node = other.head; node != nullptr; node = node->next)

{

push\_back(node->data);

}

}

return \*this;

}

void push\_back(const T& data)

{

if (head == nullptr)

{

head = new Node(data);

}

else

{

Node\* tail = head;

while (tail->next != nullptr)

{

tail = tail->next;

}

tail->next = new Node(data);

}

size++;

}

void pop\_back()

{

if (head == nullptr)

{

return;

}

if (head->next == nullptr)

{

delete head;

head = nullptr;

}

else

{

Node\* node = head;

while (node->next->next != nullptr)

{

node = node->next;

}

delete node->next;

node->next = nullptr;

}

size--;

}

int getSize() const { return size; }

T& operator[](int index)

{

Node\* node = head;

for (int i = 0; i < index; i++)

{

node = node->next;

}

return node->data;

}

const T& operator[](int index) const

{

Node\* node = head;

for (int i = 0; i < index; i++)

{

node = node->next;

}

return node->data;

}

List<T> operator\*(const List<T>& other) const

{

List<T> result;

if (size == other.size)

{

for (int i = 0; i < size; i++)

{

result.push\_back((\*this)[i] \* other[i]);

}

}

return result;

}

friend std::ostream& operator<<(std::ostream& os, const List<T>& list)

{

for (Node\* node = list.head; node != nullptr; node = node->next)

{

os << node->data << ' ';

}

return os;

}

friend std::istream& operator>>(std::istream& is, List<T>& list)

{

T data;

while (is >> data)

{

list.push\_back(data);

}

return is;

}

void clear()

{

while (head != nullptr)

{

Node\* node = head;

head = head->next;

delete node;

}

size = 0;

}

};

#endif // LIST\_H

**main.cpp**#include <iostream>

#include "list.h"

#include "pair.h"

using namespace std;

int main() {

// Тестирование списка со стандартными типами данных

List<int> intList;

intList.push\_back(1);

intList.push\_back(2);

intList.push\_back(3);

intList.push\_back(4);

cout << "List of integers: ";

for (int i = 0; i < intList.getSize(); i++) {

cout << intList[i] << " ";

}

cout << endl;

List<float> floatList;

floatList.push\_back(1.2);

floatList.push\_back(2.3);

floatList.push\_back(3.4);

floatList.push\_back(4.5);

cout << "List of floats: ";

for (int i = 0; i < floatList.getSize(); i++) {

cout << floatList[i] << " ";

}

cout << endl;

List<double> doubleList;

doubleList.push\_back(1.23);

doubleList.push\_back(2.34);

doubleList.push\_back(3.45);

doubleList.push\_back(4.56);

cout << "List of doubles: ";

for (int i = 0; i < doubleList.getSize(); i++) {

cout << doubleList[i] << " ";

}

cout << endl;

// Тестирование списка с пользовательским классом Pair

List<Pair> pairList;

pairList.push\_back(Pair(1, 1.1));

pairList.push\_back(Pair(2, 2.2));

pairList.push\_back(Pair(3, 3.3));

pairList.push\_back(Pair(4, 4.4));

cout << "List of pairs: ";

for (int i = 0; i < pairList.getSize(); i++) {

Pair p = pairList[i];

cout << "(" << p.getFirst() << ":" << p.getSecond() << ") ";

}

cout << endl;

return 0;

}